Matain-e-nator: *Make the World a Better Place*

Xin Liu
University at Buffalo
1932 Wallamaloo Lane
xliu36@buffalo.edu

John Longanecker
University at Buffalo
P.O. Box 1212
webmaster@marysville-ohio.com

Juehui Zhang
University at Buffalo
Hekla, Iceland
larst@affiliation.org

ABSTRACT
Our goal is to improve the overall quality of a facility as well as decrease an organizationÕs overall operating expenses. By letting those who maintain facilities know about problems sooner. They can react quicker and more efficiently if they have better information about the status of their facilities. *Maintain-e-nator* provides a cell phone application to report problems as well as a web interface to allow maintenance workers to be notified of new problems.

Categories and Subject Descriptors
H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms
Applications

Keywords
ACM proceedings, LaTeX, text tagging

1. INTRODUCTION
Eventually everything breaks. Nothing lasts forever. Buildings start as brand new but eventually break down. Roads start as smooth, but eventually develop potholes. These breakdowns can sometimes be ignored like a squeaky door, but others can cause safety and health risks. If the stairs of a building are in disrepair they could cause a tripping hazard for other people.

2. MOTIVATIONS
It is easy for a large organization to be unaware of all the maintenance problems that their facilities have. Some obscure room may need a light bulb replaced but those that work in that area do not report the problem or are not around when the build is exhibiting its behaviors. So the people who take a night class are the only ones aware of the problem. They do not know where to submit a problem and are often not willing to do the necessary research to find out how to report a problem.

These problems are not limited to the indoors. They can also involve roads, landscaping, sidewalks, outdoor sports facilities.

2.1 What is a problem?
We consider a maintenance problem anything that can hurt someone as well as something that detracts from the overall quality of the facilities. So a dirty floor or table could be considered a problem. At the end of the day the users who submit a problem are the ones who are deciding what a problem is. Who better to determine a problem then those who actually use the facilities?

2.2 Goal
The overall goal is to make maintenance workers aware of the problems that exists on their property. Are android application and web interface will not promise cleaner facilities. Our goal is to help those that manage a property.

3. DESIGN
Our application comprises of two part: the Android App and the backend server, which are supposed for two kind of users: `submitter` and `maintainer`. Data flow can be depicted as Fig. . . . .

3.1 Android Application
We programmed our app[1] on the Android [1] platform. Our app mainly has three modules: 1. login module 2. localization module 3. information module
Below we talk about these three modules in detail.

3.1.1 Login
Whenever some submitter wants to submit issues and open the app, she can choose to log in the app with her Google account or anonymously. We choose Google account because it is available in (or necessary for) every Android device. We add some personal decorations for user logs in with Google account, and once the user submit an issue, we will also pass their personal information (i.e., name and email address) to backend server, which maybe used in the future for contacting with the user. With the `AccountManager` in Android

[1] `https://github.com/forkloop/Maintain-e-nator`

library, we can only get the submitter's email address. To make our app more personalize, we also want the submitter's profile information. Once the submitter grant the app permission to retrieve her profile with the access token, we can use it to get the submitter information via calling User-Info Google API [3]. This whole process follows the `OAuth2` specification [5];

3.1.2 Localization

Each time when a submitter open the activity for filling an issue, the app will start to request a single location update via Network or GPS. We prefer Network over GPS for it is much faster and accurate enough with the omni Wi-Fi APs within campus. Also since it is unlikely for the submitter to move around when submitting an issue, we only request *single* location update to save battery life. If the submitter is indoor, we will try to guess which hall she maybe in by calculating the Euclidean distance between current geolocation and some predefined geolocations of halls we current support, and find the smallest one which could be the hall the submitter is in. However, if the submitter is outdoor, we will try to get a meaningful location of the submitter by using Google Place API [4] with current geolocation data.

Submitter can change the location information the app provides later. Also, for outdoor scenario, the app has a map with a marker which indicate where submitter currently is. However, if the geolocation provided is not that accurate or submitter moves around, she can adjust the position of marker, and the app will then use that new geolocation accordingly.

3.1.3 Info

For future maintainers can locate the issue position accurately and easily, we hope submitters can report the issue location as detail as possible. Since the location for indoor and outdoor could be very different, to ease the process of filling the detail of an issue, we has two different forms for indoor and outdoor separately, as Fig 1. The app requires submitter to at least provide a description, the location (which maybe acquired automatically as described in Sec. 3.1.2 or replaced by submitter). For current indoor experiment, we only support four halls - Davis Hall, Jarvis Hall, Furnas Hall and Ketter Hall.

Only text may not be that descriptive when describing an issue, thus we allow submitter to add photos and also an audio recording regarding the location or issue detail. By `long-press` the image area, the submitter can add up to three photos either from camera capture or gallery. Of course the submitter can view the full-size photo by `clicking` it and decide whether to keep it or replace it with a new one. Also, when submitter `holding` the recording button, the app will recording the audio in `wav` format. We choose `wav` because it is the quick and dirty way to enable the recorded audio can be played back both on Android and browser. The supported recording formats on Android (or to our best knowledge on Nexus S) , e.g., `mp4`, `3gp` are not supported by current browsers. While the other formats supported by browser without any plugins, `mp3, ogg` need third-party native libraries to be recorded on Android devices.

3.1.4 Others

We will also save all the issues reported by one submitter locally on the Android device if the submitter choose to. Submitter can later view the details of all the issues she reported before.

3.2 Web Application

For backend web service, we host our web application code on `heroku` [2], which is a cloud application platform that saves us from all kinds of pitfalls in servers management, deployment. Since `heroku` does not store static files as well as users upload files, we use `Amazon S3` [3] for this task.

We use `django` [2] web framework for our web application, its active community makes it much easier to develop web apps with numerous open source modules (a full list of all open source modules we used [4]). Our web application mainly has two components, private admin site, which is for maintainers to manage all the reported issues, and public site, which is for every one to view existing reported issues.

3.2.1 Private admin site

`django` is already packed with an admin module, which is not that fancy. We instead use `grappelli` [5] for polishing up the admin interface. The admin site allow multiple maintainers to manage the reported issues, filtering them based on the location, severity level, status, submitted date *etc.* Maintainers can view the issues locations on Google Maps, get an idea of what the issues are via submitted photos and audio recording. Maintainers can also record their working progress for a specific issue.

3.2.2 Public site

The public site is intended for all to browse existing issues. We work on this public site to make it like an Single-page application [6], which can provide user a better experience with waterfall-like scrolling, no more page reloading when view different issues. We use `Backbone.js` [6] as for our frontend MVC. The whole interaction with the backend is via the RESTful JSON interface.

3.2.3 Others

The web application provides a RESTful API, which the Android application uses to submit an issue, or the public site uses to retrieve the issues information. We use `tastypie` [7] to build this REST-style interface. We did little hack on `tastypie` for this API to support file upload via `mixin`.

There are two kinds of processes for our web application, the `web process` is for responding all HTTP requests, either rendering a view or return some JSON data. While the `worker process` is for all other not emergent tasks, *e.g.*, sending submitters emails.

[2] `https://www.heroku.com/`
[3] `https://aws.amazon.com/s3/`
[4] `https://github.com/forkloop/Maintainenator-backend/blob/master/requirements.txt`
[5] `http://www.grappelliproject.com/`
[6] `http://backbonejs.org/`
[7] `http://tastypieapi.org/`

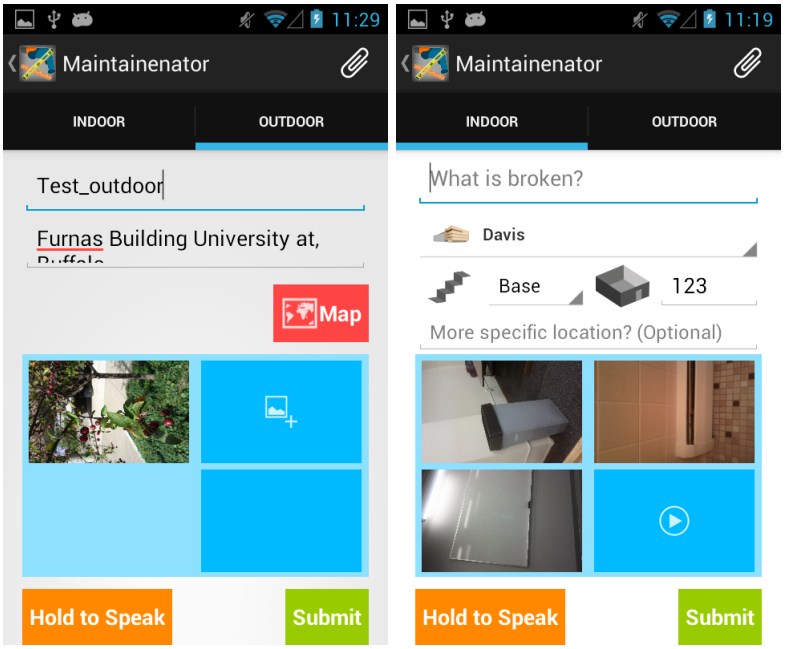

Figure 1: Issue form.

4. CONCLUSIONS

. . .

5. ACKNOWLEDGMENTS

. . .

6. REFERENCES

[1] Android. http://www.android.com/.
[2] Django. https://www.djangoproject.com/, 2012.
[3] Google. https://developers.google.com/accounts/docs/oauth2.
[4] Google-Places-API. https://developers.google.com/places/documentation/.
[5] O. W. Group. The oauth 2.0 authorization framework. 2012.
[6] Wikipedia. Single-page application — wikipedia, the